Week 11

Revision

Introduction

Algorithms and their analysis

Data structures

Specific techniques

Conclusion

How to prepare

All information is on the course homepage

http://cs.swan.ac.uk/~csoliver/Algorithms201112/index.html

• Go through all lectures (perhaps you download the slides again — certain additions have been made).
• Go through the courseworks and their solutions.
• Go through the lab sessions (run it!).

Use the learning goals!

General remarks

You need to be able to perform the following:

• Reproducing all definitions.
• Performing all computations on (arbitrary) examples.
• Explaining all algorithms in (written) words (using some mathematical formalism).
• Stating the time complexity in term of $\Theta$.

Check all this by writing it all down!

Actually understanding the definitions and algorithms helps a lot.

On the exam

We use the “two-out-of-three” format.

• Give us a chance to give you marks — write down something!
• Actually, better write down a lot — in the second or third round, after answering all questions.
• Examples are typically helpful.
The general structure of the module

- Weeks 1-3: Algorithms and their analysis
- Weeks 4-7: Data structures – graphs and trees
- Weeks 8, 9: Specific techniques – greedy algorithms and dynamic programming
- Week 10: Complexity theory – P and NP.

Order of growth

Get a working understanding of $O$, $\Omega$, $\Theta$.

This includes developing an intuitive grasp on the main types of growth rates:

- $\log n$, $\sqrt{n}$, $n$, $n \log n$, $n^2$, $n^3$, $2^n$, $n!$

- Of course, you need also to know the definitions.
- And you need to know the various techniques for working with terms ("rewrite rules").

Sorting algorithms

We considered two sorting algorithms in detail:

- Insertion Sort and Merge Sort.

These algorithms you should know in detail:

- Both algorithms are comparison-based.
- You should know precisely how the comparisons are performed by both algorithms.

Solving recurrences

Know the (simplified) master theorem!

- Especially know the three cases.
- And know various examples.

It’s mostly practice.
### Concepts

For data structures, various definitions and their relations are of central importance:

- Know the definitions of "graphs" and "digraphs".
- Know the definition of a "dag".
- Know the definition of "connected (di)graphs".
- Know the definition of a "tree".
- Know what a "rooted tree" is:
  - Many important notions are connected to it ("root, children, parent, leaf").
  - Know what "ordered rooted trees" are.
  - Know what "binary trees" are.

### Further connections to graphs: MSTs

For clarification:

When running BFS or DFS on a connected graph, we get a spanning tree.

- When considering greedy algorithms, “minimum spanning trees” were considered.
- These are special spanning trees.
- It is assumed that (additionally) every edge got a “weight” (a number).
- A minimum spanning tree then is such a spanning tree whose sum of edge weights is minimal (amongst all possible spanning trees).

### Basic graph algorithms

You need to know BFS, DFS well.

This means:

- Explain them.
- Run them.
- Know what $d[u]$ for BFS is.
- Know what $d[u], f[u]$ for DFS is.

The main application of DFS we considered is topological sorting.

### Binary search trees

Don’t mix up these trees with the spanning trees we encountered with graphs.

- Know the basics of the abstract data types considered.
- Know and understand the notion of a “binary search tree”.
- Understand searching in such trees.
- Get the basic ideas of insertion and deletion.

Central is to understand the dependency of the complexity of the operations on the height of the trees.
Disjoint sets

- Know the three basic operations (creation, finding, combination).
- Know the two basic data structures.
- Know all the heuristics, their motivations and their effects.
- Their effects concern worst-case performance of some of the basic operation.

Again:

Central is to understand the dependency of the complexity of the operations on the height of the trees.

Greedy algorithms

Be familiar with the ideas.

- Know the criterions.
- Know the examples.
- Besides making change, we considered Kruskal’s algorithm for computing MST.
  - Understand the definition of "MST".
  - Understand how disjoint sets are involved here.
  - Of course, know the algorithm.

Making change

The “Change Making” problem is a central problem in the section on specific techniques:

- You need to know its precise definition.
- You need to know the greedy algorithm for.
- You need to know the limitations of the greedy algorithms (with examples).
- Then comes the dynamic programming algorithm: Know it in detail.
- As usual, you need also to be able to run the algorithms, and show the steps of the computations.

Dynamic programming

Again, be familiar with the ideas.

- Again, know the criterions.
- Again, know the examples.
- Besides making change, we considered the Floyd-Warshall algorithm for computing the distance-matrix:
  - Understand the definition of “distance matrix”.
  - Understand the basic recursive equation.
Further remarks

- Divide-and-conquer is an important general paradigm, and there are several typical examples.
- Complexity theory underlies all of algorithmics, but it is not part of the exam.
- Try to grasp the general characteristics of algorithmical paradigms we considered (divide-and-conquer, graph-search, greedy, dynamic programming). Each is based on a fundamental “trick”.

Nearly the end

I wish you a productive preparation phase.

Tomorrow we will consider some specific exam-like questions.